Detecting Sensitive Behavior on Android with Static Taint Analysis Based on Classification

Yayun Chen[1, a], Hua Zhang[1, a]

[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[a]cyy310@bupt.edu.cn

Keywords: Classification, Sensitive Value, Static Taint Propagation.

Abstract. This paper devotes to the APK sensitive behavior was studied by classification, and quantifies the APK sensitive behavior, finally after a quantitative APK sensitive behavior can be measured on the basis of the APK sensitive value, through the unique static detection method based on classification show users the APK is possible sensitive behavior and value. First of all, the APK according to its function of different will have different behavior, but beyond the functional behavior of additional behavior belongs to sensitive behavior, these often is the core of the leak privacy sensitive behavior, thus more targeted. Secondly, the static stain were used to detect, to cover the logical path for as long as possible, can be compared with the dynamic testing gives a more detailed test results. Many researchers are from the static current detection algorithm is to find a better solution, here from another train of thought, the classification property of APK itself to trigger a new thinking.

1. Introduction

Mobile client has become the important carrier of the Internet era, Android system has obtained great development because of its open source and the characteristics of wide use. Felt etal. classified different kinds of Android malware [1] and found that one of the main threats posed by malicious Android applications are privacy violations which leak sensitive information such as location information, contact data, pictures, SMS messages, etc. to the attacker.

In existing detection tools, the representative of the dynamic detection tools is TaintDroid [2], the representative of the static tools are LeakMiner[3] and FlowDroid [4], this paper research work will be based on FlowDroid. And the point of this paper is focused on the classification, which means that the same sensitive calls will produce different effects in different types of the APK.

First we detect APK using a common configuration file, to get the APK statistics results. By studying the detection results, we find that the sensitive calls in the configuration file will not be used in all types of APK. Therefore, the work can make detection more targeted. And in the same class APK, different sensitive calls have different frequency of use. We will use the frequency to calculate the sensitive value of the call and the whole APK.

Therefore, the emphasis of the paper is not in research ICFG and CALLGRAPH, but in the classification. Load configuration file base of classification code, to make the detection more targeted.

2. Architecture

In this paper, the general structure of the system structure consists of four main modules, respectively is input, preprocessing, processing and analysis. Input module provides APK file and configuration files for the next step, preprocessing module loads APK and configuration files to be detected firstly, the processing module decompiles DEX file, fi-nally a information flow propagation will be executed to find sensitive paths in the input APK file, get the sensitive path may be. As shown in Fig. 1.

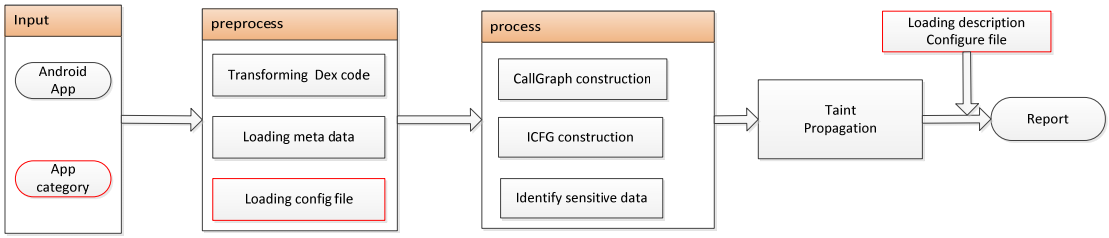

Fig. 1 Architecture

Input. Input contains two parts, the APK file and the corresponding classification code, classification refers to the function of the APK file, this paper combines many application classification on the network and proposed to a simple and accurate classification, make sure we can cover the man function of android apps. The input function code will be used to determine corresponding configuration file in the next step.

Preprocessing. The APK is essentially a zip file, so we need to extract the APK file, obtain the AndroidManifest. XML file and DEX file at first. A plugin called Dexpler [5] allows we to convert Android's Dalvik bytecode into Jimple[6].

Secondly according to the app category code, choose the corresponding sensitive call configuration file to load, these configuration files are gained through detecting a lot of an-droid apps which are classified, and record the frequently used sensitive calls of corresponding APK function type, to make detection more targeted.

Sensitive calls mainly divide into two parts, one is source call, the other one part is sink call. Source call is used to obtain sensitive information of mobile phone, such as getServiceId(), sink call will send sensitive data out , such as sendTextMessage ().

Processing. Though the CALLGRAPH of the whole APK and CFG of each method generated by soot[7], we can get the ICFG, and then according to configuration file loaded in the preprocessing module, identify sink sensitive calls.

Flow propagation. Sensitive path is a path between source of sensitive call and source sensitive call, this path will be used to describe sensitive behavior. We traverse ICFG to get the sensitive path. On top of Soot and Dexpler, We further use Heros [8, 9], a scalable, highly multi-threaded implementation of the IFDS framework [10]. We identity sensitive calls in ICFG first, and then execute a reverse traversal to obtain corresponding source calls. At last we can get a connection between two sensitive calls.

Calculating sensitive value. We will analyze sensitive behavior of these sensitive paths after we get them. Depending on the source sensitive call and sink sensitive call we identify sensitive behavior for each path, and calculate sensitive value of these paths and the whole APK.

3. Implementation

This section describes the details of detection, contains the configuration files' generating of each APK type, how to calculate sensitive value in corresponding configuration file, and the method of get sensitive value of behaviors and the whole APK.

3.1 Generating configuration files. Configuration file is a key part of detection. Each category will correspond to a configuration file, at the time of preprocessing is loaded, when processing, will be in accordance with the sink in the configuration file and the source function to traverse ICFG.

The configuration files for each APK type is generated based on the actual detected results, and there are 10 configuration files for 10 classifications at last. Each configuration file is according to the corresponding APK detected results.

Through testing a large number of APK files, statistics the use frequency for each type APK files, and calculate the percentage, these configuration data will be used in calculating sensitive value of submitted APK file.

In order to produce the required configuration file, we first configure a broad sense file, this broad sense configuration file including the network connection, sending information, access to information, access to mobile phones state, access to location information, access to installation

package information, log related operations and access to contact information. These contain the most common sensitive issue. On the other hand, we divide the type of APKs into 10 types, respectively the social communication network(1), the convenient life(2), audio and video (3), photography optimization (4), theme wallpaper (5), financing and shopping (6), system tools (7), information reading (8), tourism travel (9), office learning (10), every kind of APK file using the bread sense configuration file to test, and get the results for every APK type. Count the calls of a single APK and then get the statistics of the calls in same APK type.

As mentioned above, classifying android apps according to its use function, the main purpose is to make the detection more targeted in the detection processing. Classification detection has the following advantages: 1. Detection can focus on the main functions of APK, then will be closer to the actual situation; 2. Depending on the ratio of corresponding type of APK we can calculate the sensitive value of the whole APK file.

3.2 Calculating sensitive value. How to quantify the results of detection processing, make the results of the test is more reading and reference more can reflect the APK stands for itself, is also this article try to one direction. Excellent open source projects FlowDroid on the accuracy of the APK testing done a lot of work, but they are the test results are given in has not read, not to mention the exact embodiment of APK sensitivity problem in itself.

After getting the invocation path between the source and the sink, we will be combined with descriptive profile to get more detail information: 1, the sensitive coefficient of the APK itself, 2 the sensitive brief functional description of the path, 3 the sensitive path sensitive issues that may arise out of the 4 sensitive details of the path.

3.2.1 Calculating. Sensitivity value calculation has the following elements: *source_api ratio*, *sink_api ratio*, the two ratios is from the configuration file in 3.1, and then be used in the following formulas:

P1 := source_api ratio (1)

P2 := sink_api ratio (2)

SP := sensitive value of path (3)

$$SP = P1 * \alpha_1 + P2 * \alpha_2 \quad (4)$$

α_1 and α_2 is corresponding *coefficient* of source_api and sink_api.

The main elements of calculating APK sensitive value are path sensitive value and path power, the rules are listed as follows :

Divide sensitive path into three levels according sensitive coefficient: *high, middle, low.*

high $[h_l, h_h]$, power β_h (5)

middle $[m_l, m_h]$, power β_m (6)

low $[l_l, l_h]$, power β_l (7)

Identity the *sensitive level* of each path according sensitive path vale, and then calculate the expectation of each level, multiply by the power of each level respectively, calculate the overall expectation, get the APK sensitive value at last.

SL : sensitive value of level (8)

i : high, medium, low (9)

$$SL_i = \frac{\sum SP_i}{\sum i} \quad (10)$$

SA : sensitivity value of APK

(11)

$$SA = \frac{\sum_{i=0}^{3} SL_i}{3} \quad (12)$$

3.2.2 Identifying behavior. Through configuration files, detection report summarizes and classifies the malicious behavior, the eight types are shown in table 1, the detail description is in specification [11].A friendly report is also an important element for measuring a detection report,

through a summary the sensitive paths can be more intuitive and effective to observe a APK sensitive features.

Table 1 Malicious Types

1	payment
2	privacy
3	remote
4	spread
5	expense
6	system
7	fraud
8	rogue

4. Evaluation

Compared to the other approach which is not classified, our method's advantage is can calculate the sensitive value of the detected information leakage behavior, let users and researchers have more intuitive understanding of sensitive behavior in android app. Through a large number of tests, we find that different types of APKs have different sensitive behavior percentage.

As depicted in Fig. 2, we can see the distribution of sensitive behavior in different APKs. We also can find that there are some sensitive behavior produce larger fluctuations, the other are smaller, for the latter, we will have more study on its function in the future, to determine whether the behavior and the main function have more differences, and then introduce the correlation coefficient.

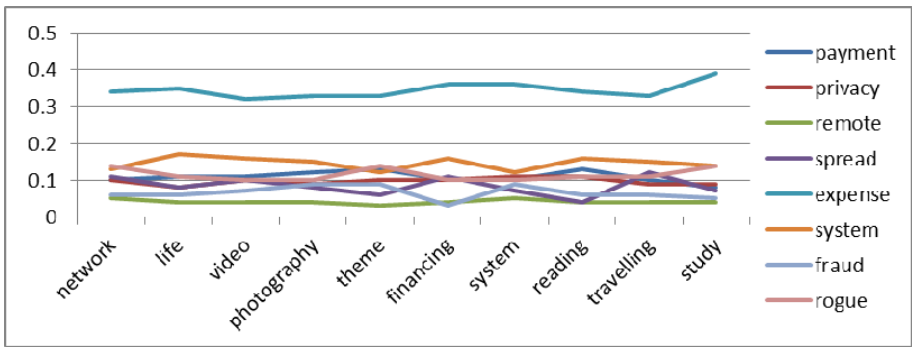

Fig. 2 Comparison of Different APKs

In table 2 shows the statistical results, the report path is detected by our tool. The true path is checked manually based on the former results.

Table 2 Report of Detection

	Report path	True path
network	278	196
life	189	180
video	300	265
photography	135	117
theme	57	34
financing	98	79
system	102	80
reading	86	69
travelling	104	91
study	79	56

5. Conclusion

In this paper, on the basis of the existing static taint analysis, proposed the solution to detect APKs according its feature, the scheme is mainly reflected in three parts: the first is the need to choose the appropriate feature type, the second is loading the configuration file depending on the type of the input file, the third is choosing the suitable ratio when calculating the detecting results.

After introducing the ideas of classification, relative to detect APK directly, its advantage is embodied in the following three points: first, to detect sensitive calls more targeted, secondly, can be sensitive to detect the path for quantitative analysis, third, carries on the quantitative analysis of time for different types of APK sensitive path can give different reference points.

References

[1] P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, A survey of mobile malware in the wild, In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, SPSM ’11, pages 3–14, New York, NY, USA, 2011. ACM.

[2] Zhemin Yang and Min Yang, LeakMiner: Detect Information Leakage on Android with Static Taint Analysis, In Software Engineering(WCSE), 2012 Third World Congress on, pages 101-104, 2012.

[3] W. Enck, P. Gilbert, B. gon Chun, L. P. Cox, J. Jung, P. McDaniel, and A, Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones, In R. H. Arpaci-Dusseau and B. Chen, editors, OSDI, pages 393–407, USENIX Association, 2010.

[4] Steven Arzt, Siegfried Rasthofer, Christian Fiz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Oceau and Patrick McDaniel, FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps, In PLDI’14.June 9-11 2014, Edinburgh, United Kingdom.

[5] Alexandre Bartel, Jacques Klein, Yves Le Taon, and Martin Monperrus, Dexpler: Converting Android DalvikBytecode to Jimple for Static Analysis with soot, In SOAP’12 June 14, Beijing, China.

[6] Raja Vallee-Rai and Laurie J.Hendren, Jimple: Simplifying Java Bytecode for Analyses and Transformations, 1998.

[7] P. Lam, E. Bodden, O. Lhotak, and L. Hendren, The soot framework for java program analysis: a retrospective, In Cetus Users and Compiler Infastructure Workshop (CETUS 2011), Oktober 2011.

[8] Heros, http://sable.github.io/heros,retrieved 2015-1-23.

[9] E. Bodden, Inter-procedural data-flow analysis with ifds/ide and soot, In Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis, SOAP ’12, pages 3–8, 2012.

[10] T. Reps, S. Horwitz, and M. Sagiv, Precise interprocedural dataflow analysis via graph reachability. In POPL ’95, pages 49–61, 1995.

[11] Internet Society of China, Anti Network-Virus Alliance of China, Specification for Mobile Internet Malicious Code. In 2011.